APPARATUS AND METHODS FOR INTELLIGENTLY CACHING APPLICATIONS AND DATA ON A GATEWAY

FIELD OF THE INVENTION

This invention relates to apparatus and methods for intelligently caching applications and data. In particular, this invention relates to apparatus and methods for intelligently caching applications and data on a gateway.

BACKGROUND OF THE INVENTION

Generally, wireless/mobile devices are connected to servers on the Internet through one or more gateways. Using a micro-browser application on a mobile device, a user may browse the Internet through the gateway(s).

Most wireless/mobile devices have inadequate processing capability for retrieving information, such as applications or data, and very limited memory space for caching such information. Thus, downloading applications or data from the Internet onto a mobile device may be very slow and sometimes unsuccessful. One possible solution to circumvent the need to repeatedly download the same applications and data from servers connected to the Internet is to cache them on the gateway(s). Gateways also have limited memory space and cannot cache all available applications and data; thus, an intelligent caching of the most likely to be called applications or data is necessary to optimize this solution.

Thus, it is desirable to provide apparatus and methods for intelligently caching applications and data on the gateway.

SUMMARY OF THE INVENTION

An exemplary method for intelligently caching applications and data on a gateway comprises the steps of calculating a cache benefit index for a set of files, the cache benefit index indicating a total benefit for caching the set of files, determining whether to cache the set of files on a local file system based on the cache benefit index, caching the set of files on the local file system, and updating a set of tables in a gateway database based on the caching.

In one embodiment, the determining step includes the steps of checking available free space in the local file system and allowing caching of the set of files into the local file system if the local file system has enough available free space for storing the set of files. In another embodiment, the determining step includes the steps of comparing the cache benefit index to a previously calculated cache benefit index for the set of files and allowing caching of the set of files if the cache benefit index is higher than the previously calculated cache benefit index. In yet another embodiment, the determining step includes the steps of comparing the cache benefit index to other cache benefit indices of files already cached on the local file system and allowing caching of the set of files if the cache benefit index is higher than the other cache benefit indices.

In an exemplary embodiment, the method further comprises the steps of recalculating a new cache benefit index for the set of files upon receiving a request to download or update the set of files and updating the set of tables in the gateway database based on the new cache benefit index.

When a download request for the set of files is received by the gateway, the exemplary method further comprises the steps of accessing the set of files in the local file system if the set of files is cached and up-to-date, creating a download response to the download request, the download response including the set of files, and sending the download response. In one embodiment, the set of files is updated from a server if it is cached but not up-to-date. In another embodiment, the set of files is downloaded from a server if it is not cached. In this embodiment, the set of files is downloaded by sending a request to the server, receiving a response from the server, the response including the set of files, parsing the response for any broadcast message, accessing and updating the gateway database if the response includes a broadcast message, and sending a broadcast response to the server.

When an update request for the set of files is received by the gateway, in one embodiment, the exemplary method further comprises the steps of accessing the local file system to obtain at least one difference file and a broadcast message if the set of files is cached and up-to-date, creating an update response to the update request, the update response including the at least one difference file and the broadcast message, and sending the update response. In another embodiment, the exemplary method further comprises the steps of downloading the set of files from a server if the set of files is not cached, creating an update response to the update request, the update response including the downloaded set of files, and sending the update response. In yet another embodiment, the exemplary method further comprises the steps of receiving at least one difference file from a server if the set of files is cached but is not up-to-date, creating an update response to the update request, the update response including the at least one difference file, and sending the update response. In yet another embodiment, the exemplary method further comprises the steps of downloading a current version of the set of files from a server if the set of files is cached but is not up-to-date, generating at least one difference file based on the current version, creating an update response to the update request, the update response including the at least one difference file, and sending the update response.

When a status check request for the set of files is received by the gateway, in one embodiment, the exemplary method further comprises the steps of accessing the local file system to load any broadcast information if the set of files is up-to-date, creating a status check response, the status check response including the broadcast information and a status of the set of files, and sending the status check response. In another embodiment, the exemplary method further comprises the steps of sending a request to a server if the set of files is cached and is not up-to-date, receiving a server response from the server, the server response including a current version and status of the set of files, updating the gateway database based on the current version and status, creating a status check response, the status check response including the status of the set of files, and sending the status check response. In an exemplary embodiment, at least one difference file is generated based on the current version, the set of files is updated based on the difference file(s), and the difference file(s) is sent in the status check response. In another exemplary embodiment, the server response is parsed for any broadcast message, the gateway database is accessed and updated if the server response includes a broadcast message, and a broadcast response is sent to the server.

In yet another embodiment, the exemplary method further comprises the steps of downloading a current version of the set of files from a server, comparing the current version of the set of files to the set of files, generating the status based on the comparing, creating a status check response, the status check response including the status of the set of files, and sending the status check response.

An exemplary computer program product for use in conjunction with a computer system for intelligently caching applications and data on a gateway comprises logic code for calculating a cache benefit index for a set of files, the cache benefit index indicating a total benefit for caching the set of files, logic code for determining whether to cache the set of files on a local file system based on the cache benefit index, logic code for caching the set of files on the local file system, and logic code for updating a set of tables in a gateway database based on the caching.

In one embodiment, the logic code for determining includes logic code for checking available free space in the local file system and logic code for allowing caching of the set of files into the local file system if the local file system has enough available free space for storing the set of files. In another embodiment, the logic code for determining includes logic code for comparing the cache benefit index to a previously calculated cache benefit index for the set of files and logic code for allowing caching of the set of files if the cache benefit index is higher than the previously calculated cache benefit index. In yet another embodiment, the logic code for determining includes logic code for comparing the cache benefit index to other cache benefit indices of files already cached on the local file system and logic code for allowing caching of the set of files if the cache benefit index is higher than the other cache benefit indices.

In an exemplary embodiment, the computer program product further comprises logic code for recalculating a new cache benefit index for the set of files upon receiving a request to download or update the set of files and logic code for updating the set of tables in the gateway database based on the new cache benefit index.

When a download request for the set of files is received by the gateway, the exemplary computer program product further comprises logic code for accessing the set of files in the local file system if the set of files is cached and up-to-date, logic code for creating a download response to the download request, the download response including the set of files, and logic code for sending the download response. In one embodiment, the computer program product includes logic code for updating the set of

files from a server if it is not up-to-date. In another embodiment, the computer program product includes logic code for downloading the set of files from a server if it is not cached. In this embodiment, the logic code for downloading includes logic code for sending a request to the server, logic code for receiving a response from the server, the response including the set of files, logic code for parsing the response for any broadcast message, logic code for accessing and updating the gateway database if the response includes a broadcast message, and logic code for sending a broadcast response to the server.

When an update request for the set of files is received by the gateway, in one embodiment, the exemplary computer program product further comprises logic code for accessing the local file system to obtain at least one difference file and a broadcast message if the set of files is cached and up-to-date, logic code for creating an update response to the update request, the update response including the at least one difference file and the broadcast message, and logic code for sending the update response. In another embodiment, the exemplary computer program product further comprises logic code for downloading the set of files from a server if the set of files is not cached, logic code for creating an update response to the update request, the update response including the downloaded set of files, and logic code for sending the update response. In yet another embodiment, the exemplary computer program product further comprises logic code for receiving at least one difference file from a server if the set of files is cached but is not up-to-date, logic code for creating an update response to the update request, the update response including the at least one difference file, and logic code for sending the update response. In yet another embodiment, the exemplary computer program product further comprises logic code for receiving an update request for the set of files, logic code for downloading a current version of the set of files from a server if the set of files is cached but is not up-to-date, logic code for generating at least one difference file based on the current version, logic code for creating an update response to the update request, the update response including the at least one difference file, and logic code for sending the update response.

When a status check request for the set of files is received by the gateway, in one embodiment, the exemplary computer program product further comprises logic code for accessing the local file system to load any broadcast information if the set of files is up-to-date, logic code for creating a status check response, the status check

response including the broadcast information and a status of the set of files, and logic code for sending the status check response. In another embodiment, the exemplary computer program product further comprises logic code for sending a request to a server if the set of files is cached and is not up-to-date, logic code for receiving a server response from the server, the server response including a current version and status of the set of files, logic code for updating the gateway database based on the current version and status, logic code for creating a status check response, the status check response including the status of the set of files, and logic code for sending the status check response. In an exemplary embodiment, the computer program product further comprises logic code for generating at least one difference file based on the current version, logic code for updating the set of files based on the difference file, and logic code for sending the difference file in the status check response. In another exemplary embodiment, the computer program product further comprises logic code for parsing the server response for any broadcast message, logic code for accessing and updating the gateway database if the response includes a broadcast message, and logic code for sending a broadcast response to the server. When a status check request for the set of files is received by the gateway, in yet another embodiment, the exemplary computer program product further comprises logic code for downloading a current version of the set of files from a server, logic code for comparing the current version of the set of files to the set of files, logic code for generating the status based on the comparing, logic code for creating a status check response, the status check response including the status of the set of files, and logic code for sending the status check response.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 schematically illustrates an exemplary system in accordance with an embodiment of the invention.

FIGURE 2 schematically illustrates an exemplary gateway in accordance with an embodiment of the invention.

FIGURE 3 schematically illustrates an exemplary two level transaction support process in accordance with an embodiment of the invention.

FIGURE 4 illustrates an exemplary application identification table in accordance with an embodiment of the invention.

FIGURE 5 illustrates an exemplary data identification table in accordance with an embodiment of the invention.

FIGURE 6 illustrates an exemplary subscriber registration table in accordance with an embodiment of the invention.

FIGURE 7 illustrates an exemplary application registration table in accordance with an embodiment of the invention.

FIGURE 8 illustrates an exemplary compression methods table in accordance with an embodiment of the invention.

FIGURE 9 illustrates an exemplary compatible (3i) server registration table in accordance with an embodiment of the invention.

FIGURE 10 illustrates an exemplary session management table in accordance with an embodiment of the invention.

FIGURE 11 illustrates an exemplary application download/update histories table in accordance with an embodiment of the invention.

FIGURE 12 illustrates an exemplary data download/update histories table in accordance with an embodiment of the invention.

FIGURE 13 illustrates an exemplary application storage table in accordance with an embodiment of the invention.

FIGURE 14 illustrates an exemplary data storage table in accordance with an embodiment of the invention.

FIGURE 15 illustrates an exemplary mobile application cache table in accordance with an embodiment of the invention.

FIGURE 16 illustrates an exemplary mobile application use table in accordance with an embodiment of the invention.

FIGURE 17 illustrates an exemplary broadcast table in accordance with an embodiment of the invention.

FIGURE 18 illustrates an exemplary configuration table in accordance with an embodiment of the invention.

FIGURE 19 illustrates an exemplary process in accordance with an embodiment of the invention.

FIGURE 20 illustrates an exemplary request type table in accordance with an embodiment of the invention.

FIGURE 21A illustrates an exemplary process to open a communication session in accordance with an embodiment of the invention.

FIGURE 21B illustrates an exemplary process to reuse a communication session in accordance with an embodiment of the invention.

FIGURES 22A-D illustrate four modes of exemplary application download processes in accordance with an embodiment of the invention.

FIGURES 23A-D illustrate an exemplary application download process in accordance with an embodiment of the invention.

FIGURES 24A-C illustrate three modes of exemplary application update processes in accordance with an embodiment of the invention.

FIGURES 25A-D illustrate an exemplary application update process in accordance with an embodiment of the invention.

FIGURES 26A-C illustrate three modes of exemplary application status check processes in accordance with an embodiment of the invention.

FIGURES 27A-D illustrate an exemplary application status check process in accordance with an embodiment of the invention.

FIGURE 28 illustrates an exemplary process for identifying differences between two application/data versions in accordance with an embodiment of the invention.

FIGURE 29 schematically illustrates exemplary smart connectivity protocol state machines in accordance with an embodiment of the invention

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 illustrates an exemplary system 100. The system 100 includes multiple servers connected to multiple gateways that service multiple mobile devices. For ease of explanation, only a representative number of servers, gateways, and mobile devices are shown in Figure 1. The system 100 includes servers 102-106, gateways 108A-108B, and mobile devices 110A-110C. In an exemplary embodiment, the server 104 is a compatible server (3i server) that is capable of differentially updating applications and data stored at the gateway 108 or the mobile device 110.

Figure 2 schematically illustrates an exemplary gateway 108 in accordance with an embodiment of the invention. The gateway 108 includes a communications interface 202 for communicating with a network, a microprocessor 204, a user interface 206, and a memory 208. In an exemplary embodiment, the user interface includes a user input device (e.g., keyboard) and an output device (e.g., screen). The memory 208 includes an operating system 210, gateway applications 212, a smart

connectivity module 214, a download/update manager 216, a gateway database 218, a local file system 226, a cache engine 228, a version calculator 230, a difference calculator 232, a request sender/receiver 234, a response sender/receiver 236, a smart connectivity protocol 238, and a communications transport protocol module 240 for adapting to different transport protocols in the network. In an exemplary embodiment, the gateway database 218 includes a set of application tables 220, a set of data tables 222, and a set of other tables 224 for storing subscriber, cache storage, and other information.

In an exemplary embodiment, the gateway applications 212 provide standard gateway functions. The request sender/receiver 234 receives requests sent by subscribing mobile devices 110 and passes the requests to the smart connectivity module 214. The smart connectivity module 214 determines whether an application or data requested for execution or access is already stored in the local file system 226 and whether a cached application or data is up-to-date. The smart connectivity module 214 sends a request to a remote server 102-106 via the download/update manager 216 to download or update the requested application or data if it is not stored in the local file system 226 or if it is out-of-date, respectively. The smart connectivity module 214 calls the cache engine 228 to intelligently determine (based on a calculated cache benefit index) whether a downloaded application/data should be cached, and if so, whether there is enough space to do so. Additionally, the smart connectivity module 214 maintains meta information (i.e., synchronization version and app/data identification information, etc.) for all cached application/data in the gateway database 218 in one or more of the tables 220-224. The smart connectivity module 214 generates a response to the request from the mobile device 110 and calls the response sender/receiver 236 to send the response to the mobile device 110.

When an application/data is downloaded, cached, or sent to a mobile device 110, all files belonging to that application/data, including the executable files, configuration files, property files, online help files, etc., are processed as a bundle.

Communications between the mobile device 110 and the gateway 108 or between the gateway 108 and a 3i server 104 are based on the smart connectivity protocol 238 that is stacked on top of the communication transport and protocol 240 (e.g., wireless application protocol (WAP), TCP/IP, HTTP, infra-red data association (IrDA), or Bluetooth). Communications between the gateway 108 and other servers (non-3i servers) 102 or 106 are based only on the communication transport and

protocol 240. When downloading from the server 102 or 106, the downloaded application/data needs to be further processed at the gateway 108. For example, the smart connectivity module 214 calls the version calculator 230 to generate version information regarding a downloaded application/data and calls the difference calculator 232 to generate one or more difference files by comparing a downloaded application/data to a cached application/data. Difference files are used to update cached applications/data in the local file system 226 of the gateway and/or the mobile device 110 differentially.

Figure 3 illustrates an exemplary transaction and sub-transaction management in accordance with an embodiment of the invention. During each application/data downloading or application/data caching, the smart connectivity module 214 maintains the consistency and integrity among database operations and application/data cache space management. A transaction corresponding to an application/data update or status check is created after the smart connectivity module 214 initiates the update or status check request on the mobile device 110. The transaction is committed when the smart connectivity module 214 succeeds in the update or status check processes; otherwise, if the smart connectivity module 214 fails in the processes, the transaction is rolled back to its original state. In an exemplary embodiment, during a transaction processing, the smart connectivity module 214 may also create several sub-transactions within the transaction for various database operations. For example, the sub-transactions include an application or data cache space management transaction and communication transactions with the gateway 108 or a remote server 102. Sub-transactions become fully committed when the initial transaction becomes committed.

In an exemplary embodiment, the gateway database 218 includes a number of tables 220-224. Each table is designed to maintain a type of logical information. The smart connectivity module 214 updates the gateway database 218 and the local file system 226 in accordance with each operation performed. In an exemplary embodiment, the gateway database 218 is managed in the gateway 108 by a third-party (commercially available) database management system running on one or more UNIX servers. In one embodiment, fifteen tables are maintained in the gateway database 218. Exemplary tables are illustrated in Figures 4-18 below.

Figure 4 illustrates an exemplary application identification table. The purpose of this table is to associate each application uniform resource locator (URL) to a unique identification.

Figure 5 illustrates an exemplary data identification table. The purpose of this table is to associate each data URL to a unique identifier.

Figure 6 illustrates an exemplary subscriber registration table. The purpose of this table is to maintain service subscriber registration and access control.

Figure 7 illustrates an exemplary application registration table. The purpose of this table is to maintain application registration and access control on applications.

Figure 8 illustrates an exemplary compression methods table. The purpose of this table is to associate each data compression method name to a unique identifier.

Figure 9 illustrates an exemplary compatible (3i) server registration table. The purpose of this table is to maintain a list of all 3i servers in the system 100.

Figure 10 illustrates an exemplary session management table. The purpose of this table is to maintain the properties of all live or reusable sessions.

Figure 11 illustrates an exemplary application download/update table. The purpose of this table is to track the download and update histories of all applications ever downloaded by each subscribing mobile device 110.

Figure 12 illustrates an exemplary data download/update table. The purpose of this table is to track the download and update histories of all data ever donwloaded by each subscribing mobile device 110.

Figure 13 illustrates an exemplary application storage table. The purpose of this table is to maintain the meta information associated with all cached applications in each gateway 108.

Figure 14 illustrates an exemplary data storage table. The purpose of this table is to maintain the meta information associated with all cached data in each gateway 108.

Figure 15 illustrates an exemplary mobile application cache table. The purpose of this table is to maintain a list of applications currently cached at each subscribing mobile device 110.

Figure 16 illustrates an exemplary mobile application use table. The purpose of this table is to maintain application use histories by subscribing mobile devices.

Figure 17 illustrates an exemplary broadcast table. The purpose of this table is to maintain application broadcast messages that should be piggybacked to mobile devices 110.

Figure 18 illustrates an exemplary configuration table. The purpose of this table is to set and maintain a set of configuration parameters that control the behavior of the gateway 108.

When the cache engine 228 is called, a cache benefit index (CBI) is calculated to determine if a downloaded application or data should be cached in the local file system 226. Generally, the CBI represents the total traffic volume saved in bytes between a remote server 102-106 and the gateway 108 if an application or data is cached on the gateway 108. Thus, the greater the CBI, the greater total traffic volume saved and the more benefit for caching an application or data. Calculating the CBI for each requested application or data ensures intelligent application/data caching on a gateway 108, such that an optimal set of applications and data is cached in the limited local file system space to maximize traffic reduction between the gateway 108 and remote servers 102-106.

The CBI associated with each application or data is dynamically calculated. When an application or data is requested for download or update by a subscribing mobile device 110, the CBI associated with that application or data is calculated or recalculated, respectively. Typically, CBI calculations take into account these parameters: the last application execution or data access time stamp, the application or data size, the frequency of application or data downloads and updates in mobile devices 110, an average update rate for the application or data cached in mobile devices 110, the frequency of application or data updates in the gateway 108, the average update rate of application or data cached at the gateway 108, and/or other parameters.

Typically, an application/data download request is initiated by a user at a mobile device 110. After the application/data is downloaded from a server 102-106 and cached in the local file system 226 of the gateway 108, the volume of traffic between the server 102-106 and the gateway 108 for purposes of downloading that application or data becomes zero; thus, caching an application/ data in the local file system 226 at the gateway 108 reduces traffic. Once an application/data is cached, the volume of traffic between the server 102-106 and the gateway 108 for purposes of updating that cached application/data increases; thus, the need to update a cached application or data in the local file system 226 at the gateway 108 increases traffic.

In an exemplary embodiment, "t_l" represents the last application execution or data access time stamp, "t_n" represents the current time stamp, and EFFECT_PERIOD

is as defined in the configuration table (see Figure 18). In one embodiment, any record in the application/data storage tables (see Figures 13-14) whose last execution or access time stamp (t_l) is less than or equal to t_n - EFFECT_PERIOD and having a CBI = 0 can be deleted from those tables.

For applications or data whose last execution or access is greater than t_n - EFFECT_PERIOD, their CBIs are calculated each time a user request is received to execute or access the applications or data.

For an application/data downloaded from the gateway 108 to the mobile device 110, the current number of downloads is equal to the last number of downloads plus one: $nDownload^{sm}_{new} = nDownload^{sm}_{old}+1$. Similarly, if a cached application/data in the mobile device 110 receives an update from the gateway 108, the current number of updates is equal to the last number of updates plus one: $nUpdate^{sm}_{new} = nUpdate^{sm}_{old}+1$. The new update rate, $updateRate^{sm}_{new}$, is the average update rate after the current update. The $rate^{sm}_{new}$ is the current update traffic volume divided by the application or data size multiplied by 100. The $updateRate^{sm}_{new}$ can be calculated based on the old update rate ($updateRate^{sm}_{old}$), the last number of updates ($nUpdate^{sm}_{old}$), the current number of updates ($nUpdate^{sm}_{new}$), and the $rate^{sm}_{new}$ in the following equation:

$$updateRate^{sm}_{new} = (updateRate^{sm}_{old} * nUpdate^{sm}_{old} + rate^{sm}_{new}) / nUpdate^{sm}_{new}$$

For an application/data updated from a server 102-106 to the gateway 108, the number of current updates is equal to the last number of updates plus one: $nUpdate^{sg}_{new} = nUpdate^{sg}_{old}+1$. The new update rate, $updateRate^{sg}_{new}$, is the average update rate after the current update. The $rate^{sg}_{new}$ is the current update traffic volume divided by the application or data size multiplied by 100. The $updateRate^{sg}_{new}$ can be calculated based on the old update rate ($updateRate^{sg}_{old}$), the last number of updates ($nUpdate^{sg}_{old}$), the current number of updates ($nUpdate^{sm}_{new}$), and the $rate^{sg}_{new}$ in the following equation:

$$updateRate^{sg}_{new} = (updateRate^{sg}_{old} * nUpdate^{sg}_{old} + rate^{sg}_{new}) / nUpdate^{sg}_{new}$$

Based on the discussion above, the CBI associated with an application or data can be calculated using the following equation: CBI = total download cost (TDC)- total update cost (TUC). For an application, TDC = $nDownload^{sm}$ x appSize + $nUpdate^{sm}$ x $updateRate^{sm}$ x appSize/100 and TUC = $nUpdate^{sg}$ x appSize x

updateRate^{us}/100. For data, TDC = nDownload^{gm} x dataSize + nUpdate^{gm} x updateRate^{gm} x dataSize/100 and TUC = nUpdate^{us} x dataSize x updateRate^{us}/100. As shown, the greater the TDC, the more benefit for caching an application or data; the greater the TUC, the less benefit for caching an application or data.

Figure 19 illustrates an exemplary process in accordance with an embodiment of the invention. At step 1902, a request to download or update an application/data is received by the gateway 108. The requested application/data is downloaded/updated ("current application/data") (step 1903). The CBI for the current application/data is calculated/ recalculated ("calculated CBI") (step 1904). Next, the calculated CBI is compared to an old CBI, if any, for the current application/data (step 1906). Note that the old CBI is equal to zero when the application/data was first downloaded. If the calculated CBI is greater than the old CBI (step 1908), whether the current application/data should be cached is determined (step 1910). If the calculated CBI is less than or equal to the old CBI (step 1908), the process ends.

In an exemplary embodiment, whether there is enough free space in the local file system 226 to store the current application/data is determined (step 1912). If there is enough free space in the local file system 226 to cache the current application/data, the current application/data is cached (step 1922) and available cache space in the local file system 226 is appropriately adjusted (step 1924). In an exemplary embodiment, after the current application/data is cached, the total free space in the local file system 226 is decreased by the size of the current application/data. For example, if "FM_c" represents the free space in the local file system 226, "SZ_{ad}" represents the actual size of the current application/data, and "SZ_{mi}" represents the size of the current application/data's meta information, then if $FM_c \geq SZ_{ad}+SZ_{mi}$, there is enough free space to cache the current application/data. In an exemplary embodiment, after caching the current application/data, a new total free space in the local file system is calculated as follows: $FM_{cnew} = FM_c - (SZ_{ad}+SZ_{mi})$.

Referring back to step 1912, if there is not enough free space in the local file system 226 to cache the current application/data, the calculated CBI is compared to CBIs of other applications/data cached in the local file system 226 to determine whether the current application/data should be cached (step 1914). If the calculated CBI is not greater than the other CBIs, the process ends and the current application/data is not cached in the local file system 226. If the calculated CBI is greater than the other CBIs, whether removal of some or all of the cached

application/data from the local file system 226 will generate enough free space is determined. In an exemplary embodiment, only cached applications/data having a CBI less than the calculated CBI should be potentially removable. In other words, cached applications/data having a CBI greater than or equal to the calculated CBI should not be removed. If removal of some or all of such applications/data (with smaller CBI than calculated CBI) can generate enough space to cache the current application/data (step 1918), the minimum number of such applications/data is removed to generate just enough space for the current application/data (step 1920). In an exemplary embodiment, when there are "n" cached application/data that have a CBI less than the calculated CBI, the following equations are solved:

$$\text{Minimize} \sum_{qi=1}^{m} CBI^{qi}$$

$$\text{Where} \sum_{qi=1}^{m} CBI^{qi} < CBI^{0},$$

$$FM_c + \sum(SZ_{ad}{}^{qi} + SZ_{mi}{}^{qi}) \geq SZ_{ad}{}^{0} + SZ_{mi}{}^{0}$$

where the CBI of the i^{th} ($1 \leq i \leq n$) application/data is represented as "CBI^i", the actual size of the i^{th} application/ data is represented by "$SZ_{ad}{}^i$," the size of the i^{th} application/data's meta information is represented by "$SZ_{mi}{}^i$," $m \leq n$, $1 \leq qi \leq n$, CBI^{qi}, $SZ_{ad}{}^{qi}$, $SZ_{mi}{}^{qi}$ represent the CBI, SZ_{ad}, SZ_{mi} of the qi^{th} application or data, respectively, and CBI^0, $SZ_{ad}{}^0$, $SZ_{mi}{}^0$ represent the CBI, SZ_{ad}, SZ_{mi} of the current application or data, respectively.

By solving the above equations, the smallest necessary number of cached application/data with their respective CBIs less than the current CBI is removed from the local file system 226 to accommodate the caching of the current application/data.

After the removal of some or all of the cached application/data (that generates enough space to cache the current application/data), the current application/data is cached (step 1922) and the total available space in the local file system 226 is accordingly adjusted (step 1924). In an exemplary embodiment, the new total available space is calculated as follows:

$$FM_{cnew} = FM_c + \sum^{m}(SZ_{ad}{}^{qi} + SZ_{mi}{}^{qi}) - (SZ_{ad}{}^{0} + SZ_{mi}{}^{0})$$

Referring back to step 1918, if the removal of some or all of the cached application/data still does not generate enough space to cache the current application/data, the current application/data is not cached.

Generally, operations for intelligently caching application and data on the gateway 108 are driven by requests from subscribing mobile devices 110. Figure 20 illustrates a table listing exemplary requests from the mobile devices 110. The smart connectivity module 214 processes each request received from the mobile devices 110.

Figure 21A illustrates an exemplary process when an open session request is received in accordance with an embodiment of the invention. At step 2102, an open session request is received at the gateway 108. In response to the request, the subscriber is authenticated (step 2104). In an exemplary embodiment, a subscriber ID is identified from the open session request, then the subscriber registration table (see Figure 6) in the gateway database 218 is searched for a record associated with the subscriber ID. If such a record exists, the subscriber is authenticated (step 2106); otherwise, the subscriber cannot be authenticated and an error response is generated (step 2108). Next, if the subscriber is authenticated, the smart connectivity module 214 proceeds to agree (or disagree) to a proposed capability of the subscriber in the open session request (step 2110). For example, the subscriber may have proposed a particular data compression method. In this case, the compression method registration table (see Figure 8) is searched in the gateway database 218 for a record that matches the proposed compression method. If the search is not successful, no compression method will be used for the current session. If the search is successful, the proposed compression method is selected as the compression method for the current session.

Next, a new session ID is assigned to the current session (step 2112). The smart connectivity module 214 increases the LAST_SESSION_ID property value in the configuration table (see Figure 18) by 1 and assigns that value as the session ID for the current session, and updates the configuration table to reflect the new LAST_SESSION_ID property value.

The assigned new session ID is associated with session parameters and a lifetime value (step 2114). In an exemplary embodiment, the smart connectivity module 214 reviews the SESSION_TTL property value in the configuration table. The SESSION_TTL property value represents the lifetime of the current session in

seconds. Next, session information is registered into the gateway database 218 until the session's lifetime expires (step 2116). In an exemplary embodiment, the smart connectivity module 214 inserts a new record into the session management table (see Figure 10). In one embodiment, the new record includes the session ID, the subscriber ID, the data compression method ID, a starting time stamp of the session, the smart connectivity protocol 238 version, and other session property values. A success response is created (step 2118) and sent to the mobile device 110 to indicate that a session has been opened (step 2120).

Figure 21B illustrates an exemplary process when a reuse session request is received by the gateway 108 in accordance with an embodiment of the invention. At step 2122, a session reuse request is received by the gateway 108 from a mobile device 110. The session reuse request is authenticated (step 2124). In an exemplary embodiment, the session ID is identified from the session reuse request and the session management table (see Figure 10) is searched for a matching record based on the session ID. If a matching record exists and the session lifetime has not expired, the reuse session is authenticated (step 2126). Otherwise, authentication failed and an error response is created and sent to the mobile device 110 (step 2128).

At step 2130, the subscriber ID is identified from the reuse session request and the subscriber registration table (see Figure 6) is searched for a matching record based on the subscriber ID. If a matching record exists, the subscriber is authenticated (step 2132). Otherwise, authentication failed and an error response is created and sent to the mobile device 110 (step 2128).

Next, session capabilities are recovered (step 2134). In an exemplary embodiment, the smart connectivity module 214 caches identified session information in a working memory. A success response is generated (step 2136) and sent in a reuse session response to the mobile device 110 (step 2138). In an exemplary embodiment, the session capability information is included in the reuse session response so that both the mobile device 110 and the gateway 108 can recover a session created previously.

Figures 22A-D schematically illustrate four modes of exemplary application download processes in accordance with an embodiment of the invention. In Figure 22A, the gateway 108 downloads an application/data from a server 102-106 and then downloads the application/data to the mobile device 110 without caching the application/data. In Figure 22B, the gateway 108 downloads an application/data from a server 102-106 and caches the application/data without downloading it at the mobile

device 110. In Figure 22C, an application/data is already cached on the gateway 108 and the cached version is up-to-date; thus, the application/data is downloaded from the gateway 108 to the mobile device 110 directly. In Figure 22D, an application/data is already cached on the gateway 108. However, the cached application/data is out-of-date, so the gateway 108 first updates the cached application/data from the server 102-106 then downloads the updated application/data to the mobile device 110.

Figures 23A-D illustrate an exemplary download process in accordance with an embodiment of the invention. At step 2302, an application download request is received from a mobile device 110. The application requested is authenticated (step 2304). In an exemplary embodiment, an application URL is identified from the application download request. The application identification table (see Figure 4) and the application registration table (see Figure 7) in the gateway database 218 are searched using the application URL. If a matching record exists and the associated application is not disabled (e.g., the flagSet field in the application registration table is "off"), then the application is authenticated. If the application is not authenticated, an error response is created (step 2308) and sent to the requesting mobile device 110. If the application is authenticated (step 2306), then whether the application is already cached and up-to-date is checked (step 2310). If the application is not cached (step 2312), whether a server, where the application can be downloaded, is a 3i server is determined (step 2314). If the server is a 3i server, the process continues in Figure 23B. Otherwise, the process continues in Figure 23C.

Referring back to step 2312, if the application is already cached, whether the application is up-to-date is determined (step 2316). If the application is not up-to-date, the process continues in Figure 23D. If the application is up-to-date, the application download/update histories table (see Figure 11) is updated (step 2318). Next, the CBI for this application is recalculated (step 2320). In an exemplary embodiment, the recalculated CBI is equal to the old CBI plus the application size. The application storage table (see Figure 13) is updated by replacing the old CBI with the recalculated CBI (step 2322). The local file system 226 is accessed to load the cached application (step 2324). The gateway database 218 is accessed to load meta information related to the application (step 2326). In an exemplary embodiment, the application registration table (see Figure 7) in the gateway database 218 is searched for meta information associated with the application. Next, the gateway database 218 is accessed to load broadcast information, if any (step 2328). A success response is generated (step 2330)

and an application download response is sent to the requesting mobile device 110 (step 2332). In an exemplary embodiment, the application download response includes application contents, application meta information, and broadcast message, if any.

Figure 23B illustrates an exemplary application download process from a 3i server 104 in accordance with an embodiment of the invention. At step 2334, an open/reuse communication session request is sent to the 3i server 104. An open/reuse communication session response is received (step 2336). Next, an application download request is sent to the server 104 (step 2338) and an application download response is received from the server 106 (step 2340). In an exemplary embodiment, the response includes the requested application. The response is parsed to determine whether a broadcast message is piggybacked (step 2342). If no broadcast message is piggybacked (step 2344), the process continues at step 2352. If a broadcast message is piggybacked (step 2344), the application storage table (see Figure 13) is accessed and updated (step 2346). In an exemplary embodiment, a broadcast message includes an application URL and an application version for each of one or more applications. The application storage table is searched for the *appVer* and *flagSet* fields of each record that is associated with an application URL component in the broadcast message. The *appVer* of a matching record and an application version component in the broadcast message are compared. If the versions are different, then set a corresponding *flagSet* field to indicate that the associated application is out-of-date. This process repeats for all applications in the broadcast message. Next, the mobile application cache table (see Figure 15) and the broadcast table (see Figure 17) are accessed (step 2348). In an exemplary embodiment, the mobile application cache table is searched for the "subId" field of all records that are associated with the application URLs in the broadcast message. For every identified subId field, a record comprising the subId, the application URL, and the application version is created and inserted into the broadcast table. Next, a broadcast response is sent back to the 3i server 104 (step 2350).

At step 2352, a close session request is sent to the server 104 and the communication is disconnected. The application download/update histories table (see Figure 11) and the application registration table (see Figure 7) are updated (step 2354). In an exemplary embodiment, a new record corresponding to the downloaded application is created and inserted into the application download/update histories table. In another exemplary embodiment, an existing record corresponding to the downloaded application in the application download/update histories table is updated

as follows: the "appSize" field value is replaced with the application size of the downloaded application, the "nDownload" field is incremented by 1, and the "timestamp" field value is replaced by the current time. A corresponding record in the application registration table is updated as follows: the "appVer" field is replaced with the version of the downloaded application. Next, the CBI for the downloaded application is calculated (step 2356). In an exemplary embodiment, the CBI is equal to the TDC. Whether the downloaded application should be cached is determined based on the calculated CBI (step 2358). If not, the process continues at step 2328 in Figure 23A. If the downloaded application is to be cached (step 2360), the application is cached in the local file system 226 at the gateway 108 (step 2362). Next, the application storage table (see Figure 13) is updated (step 2364) and the process continues at step 2328 in Figure 23A. In an exemplary embodiment, a new record corresponding to the downloaded application is created and inserted into the application storage table.

Figure 23C illustrates an exemplary application download process from a non-3i server 102 or 106 in accordance with an embodiment of the invention. At step 2366, a connection with the server 102 is established. Next, an application download request is sent, a response is received, and the application is downloaded from the remote server 102 (step 2368). Application meta information (e.g., application version, file versions, number of files, application size, etc.) associated with the downloaded application is generated (step 2370). The communication is disconnected (step 2372). The application download/update histories table (see Figure 11) and the application registration table (see Figure 7) are updated (step 2374). In an exemplary embodiment, a new record corresponding to the downloaded application is created and inserted into the application download/update histories table. In another exemplary embodiment, an existing record corresponding to the downloaded application in the application download/update histories table is updated as follows: the "appSize" field value is replaced with the application size of the downloaded application, the "nDownload" field is incremented by 1, and the "timestamp" field value is replaced by the current time. A corresponding record in the application registration table is updated as follows: the "appVer" field is replaced with the version of the downloaded application. Next, the CBI for the downloaded application is calculated (step 2376). In an exemplary embodiment, the CBI is equal to the TDC. Whether the downloaded application should be cached is determined based on the calculated CBI (step 2378).

If not, the process continues at step 2328 in Figure 23A. If the downloaded application is to be cached (step 2380), the application is cached in the local file system 226 at the gateway 108 (step 2382). Next, the application storage table (see Figure 13) is updated (step 2384) and the process continues at step 2328 in Figure 23A. In an exemplary embodiment, a new record corresponding to the downloaded application is created and inserted into the application storage table.

Figure 23D illustrates an exemplary application update process from the 3i server 104 in accordance with an embodiment of the invention. At step 2385, an open/reuse communication session request is sent to the 3i server 104. An open/reuse communication session response is received (step 2386). Next, an application update request is sent to the server 104 (step 2387) and an application update response is received from the server 106 (step 2388). In an exemplary embodiment, the update response includes at least one difference file for updating the application differentially. The response is parsed to determine whether a broadcast message is piggybacked (step 2389). If no broadcast message is piggybacked (step 2390), the process continues at step 2394. If a broadcast message is piggybacked (step 2390), the application storage table (see Figure 13) is accessed and updated (step 2391). In an exemplary embodiment, a broadcast message includes an application URL and an application version for each of one or more applications. The application storage table is searched for the *appVer* and *flagSet* fields of each record that is associated with an application URL component in the broadcast message. The *appVer* of a matching record and an application version component in the broadcast message are compared. If the versions are different, then set a corresponding *flagSet* field to indicate that the associated application is out-of-date. This process repeats for all applications in the broadcast message. Next, the mobile application cache table (see Figure 15) and the broadcast table (see Figure 17) are accessed (step 2392). In an exemplary embodiment, the mobile application cache table is searched for the "subId" field of all records that are associated with the application URLs in the broadcast message. For every identified subId field, a record comprising the subId, the application URL, and the application version are created and inserted into the broadcast table. Next, a broadcast response is sent back to the 3i server 104 (step 2393).

At step 2394, a close session request is sent to the server 104 and the communication is disconnected. The application download/update histories table (see Figure 11) and the application registration table (see Figure 7) are updated (step 2395).

In an exemplary embodiment, a corresponding record in the application download/update histories table is updated as follows: the "appSize" field value is replaced with the application size of the downloaded application, the "nDownload" field is incremented by 1, and the "timestamp" field value is replaced by the current time. A corresponding record in the application registration table is updated as follows: the "appVer" field is replaced with the version of the downloaded application. Next, the application cached in the gateway 108 is updated based on the at least one difference file (step 2396). A new CBI is calculated for the updated application (step 2397). In an exemplary embodiment, the new CBI is equal to the old CBI plus the updated application size minus the size difference between the updated application and the original application (diffSize). In another exemplary embodiment, the "nUpdate" field and the "updateRate" field are also updated, where the nUpdate field is incremented by 1 and the updateRate field is equal to: (old $updateRate^{sg}$ x old $nUpdate^{sg}$ + diffSize x 100/ new appSize)/ new $nUpdate^{sg}$, where new appSize is the size of the updated application.

Next, the application storage table is updated (step 2398). In an exemplary embodiment, a record associated with the updated application in the application storage table is updated as follows: the "nFile," "fNames," "fVers," "nextRel," "lang," "flagSet," "nUpdate," "updateRate," "CBI," and "updateItvl" fields are updated with new values. In an exemplary embodiment, a new updateItvl is equal to (1 - UPDATE_TM_WEIGHT) x old updateItvl + UPDATE_TM_WEIGHT x ($timestamp_{now}$ - lastUpdate), where UPDATE_TM_WEIGHT is as defined in Table 18, the lastUpdate is equal to the time of the last update, and the $timestamp_{now}$ is the current time. Next, the application is loaded from the local file system 226 on the gateway 108 (step 2399), the associated meta information is loaded from the application storage table in the gateway database 218 (step 2400), and the process continues at step 2328 in Figure 23A.

Figures 24A-24C schematically illustrate three modes of exemplary application update processes in accordance with an embodiment of the invention. In Figure 24A, the gateway 108 facilitates a direct application/data update between a server 102-106 and the mobile device 110 without caching the application/data. In Figure 24B, a requested application is already cached at the gateway 108 and the application is up-to-date; the gateway 108 directly updates the application at the mobile device 110 without contacting the server 102-106. In Figure 24C, a requested application is

already cached at the gateway 108 and the cached version is out-of-date; the gateway 108 updates the cached application with the server 102-106 then updates the application at the mobile device 110.

Figure 25A-D illustrate an exemplary application update process in accordance with an embodiment of the invention. At step 2502, an application update request is received. The application requested is authenticated (step 2504). In an exemplary embodiment, an application URL is identified from the application update request and the application identification table (see Figure 4) and the application registration table (see Figure 7) in the gateway database 218 are searched using the application URL. If a matching record exists and the associated application is not disabled (e.g., the flagSet field in the application registration table is "off"), then the application is authenticated. If the application is not authenticated, an error response is created (step 2508) and sent to the requesting mobile device 110. If the application is authenticated (step 2506), then whether the application is already cached is checked (step 2510). If the application is not cached (step 2512), whether a server, where the application can be downloaded, is a 3i server is determined (step 2514). If the server is a 3i server, the process continues in Figure 25B. Otherwise, the process continues in Figure 25C.

Referring back to step 2512, if the application is already cached, whether the application is up-to-date is determined (step 2516). If the application is not up-to-date, the process continues in Figure 25D. If the application is up-to-date, the application download/update histories table (see Figure 11) is updated (step 2518). In an exemplary embodiment, a new updateRate field is calculated (e.g., new updateRate = (old updateRate x old nUpdate + diffSize x 100/ appSize) and the "timestamp" field is updated with the current time stamp. Next, the CBI for this application is recalculated (step 2520). The application storage table (see Figure 13) is updated by replacing the old CBI with the recalculated CBI (step 2522). In an exemplary embodiment, a new CBI is equal to the old CBI- diffSize, where diffSize is the size difference between the new and old versions of the application. The local file system 226 is accessed to load the difference (i.e., at least one difference file) between the updated application and the original application at the mobile device 110 (step 2524). The gateway database 218 is accessed to load meta information related to the updated application (step 2526). In an exemplary embodiment, the application registration table (see Figure 7) in the gateway database 218 is searched for meta information associated with the updated application. Next, the gateway database 218 is accessed to load broadcast information,

if any (step 2528). A success response is generated (step 2530) and an application update response is sent to the requesting mobile device 110 (step 2532). In an exemplary embodiment, the application update response includes at least one difference file, application meta information, and broadcast message, if any.

Figure 25B illustrates an exemplary application update process when the server is a 3i server 104 in accordance with an embodiment of the invention. At step 2534, an open/reuse communication session request is sent to the 3i server 104. An open/reuse communication session response is received (step 2536). Next, an application update request is sent to the server 104 (step 2538) and an application update response is received from the server 104 (step 2540). In an exemplary embodiment, the response includes at least one difference file for updating the application. The response is parsed to determine whether a broadcast message is piggybacked (step 2542). If no broadcast message is piggybacked (step 2544), the process continues at step 2552. If a broadcast message is piggybacked (step 2544), the application storage table (see Figure 13) is accessed and updated (step 2546). In an exemplary embodiment, a broadcast message includes an application URL and an application version for each of one or more applications. The application storage table is searched for the *appVer* and *flagSet* fields of each record that is associated with an application URL component in the broadcast message. The *appVer* of a matching record and an application version component in the broadcast message are compared. If the versions are different, then set a corresponding *flagSet* field to indicate that the associated application is out-of-date. The process repeats for all applications in the broadcast message. Next, the mobile application cache table (see Figure 15) and the broadcast table (see Figure 17) are accessed (step 2548). In an exemplary embodiment, the mobile application cache table is searched for the "subId" fields of all records that are associated with the application URLs in the broadcast message. For every identified subId field, a record comprising the subId, the application URL, and the application version are created and inserted into the broadcast table. Next, a broadcast response is sent back to the 3i server 104 (step 2550).

At step 2552, a close session request is sent to the server 104 and the communication is disconnected. The application download/update histories table (see Figure 11) and the application registration table (see Figure 7) are updated (step 2554) and the process continues at step 2528 in Figure 25A. In an exemplary embodiment, a corresponding record in the application download/update histories table is updated as

follows: the "appSize" field value is replaced with the application size of the updated application, the "nUpdate" field is incremented by 1, and the "timestamp" field value is replaced by the current time, and a new updateRate is calculated. In an exemplary embodiment, new updateRate is equal to (the old updateRate x old nUpdate + diffSize x 100/appSize)/new nUpdate, where diffSize is the size difference between the new and old versions of the application. A corresponding record in the application registration table is updated as follows: the "appVer" field is replaced with the version of the updated application.

Figure 25C illustrates an exemplary application update process when the server is a non-3i server 102 or 106 in accordance with an embodiment of the invention. At step 2556, a connection with the server 102 is established. Next, an application download request is sent, a response is received, and the application is downloaded from the server 102 (step 2558). Application meta information (e.g., application version, file versions, number of files, application size, etc.) associated with the downloaded application is generated (step 2560). The communication is disconnected (step 2562).

The application download/update histories table (see Figure 11) and the application registration table (see Figure 7) are updated (step 2564). In an exemplary embodiment, a corresponding record in the application download/update histories table is updated as follows: the "appSize" field value is replaced with the application size of the downloaded application, the "nDownload" field is incremented by 1, and the "timestamp" field value is replaced by the current time. A corresponding record in the application registration table is updated as follows: the "appVer" field is replaced with the version of the downloaded application. Next, a new CBI for the downloaded application is calculated (step 2566). In an exemplary embodiment, the new CBI is equal to the TDC. Whether the downloaded application should be cached is determined based on the calculated CBI (step 2568). If not, the process continues at step 2528 in Figure 25A. If the downloaded application is to be cached (step 2570), the application is cached in the local file system 226 at the gateway 108 (step 2572). Next, the application storage table (see Figure 13) is updated (step 2574) and the process continues at step 2528 in Figure 25A. In an exemplary embodiment, a new record corresponding to the downloaded application is created and inserted into the application storage table. In an exemplary embodiment, the downloaded application is sent to the mobile device 110 in its entirety.

Figure 25D illustrates another exemplary update process from a 3i server 104 in accordance with an embodiment of the invention. At step 2575, an open/reuse communication session request is sent to the 3i server 104. An open/reuse communication session response is received (step 2576). Next, an application update request is sent to the server 104 (step 2577) and an application update response is received from the server 106 (step 2578). In an exemplary embodiment, the response includes at least one difference file for updating the application differentially. The response is parsed to determine whether a broadcast message is piggybacked (step 2579). If no broadcast message is piggybacked (step 2580), the process continues at step 2584. If a broadcast message is piggybacked (step 2580), the application storage table (see Figure 13) is accessed and updated (step 2581). In an exemplary embodiment, a broadcast message includes an application URL and an application version for each of one or more applications. The application storage table is searched for the *appVer* and *flagSet* fields of each record that is associated with an application URL component in the broadcast message. The *appVer* of a matching record and an application version component in the broadcast message are compared. If the versions are different, then set a corresponding *flagSet* field to indicate that the associated application is out-of-date. The process repeats for all applications in the broadcast message. Next, the mobile application cache table (see Figure 15) and the broadcast table (see Figure 17) are accessed (step 2582). In an exemplary embodiment, the mobile application cache table is searched for the "subId" fields of all records that are associated with the application URLs in the broadcast message. For every identified subId field, a record comprising the subId, the application URL, and the application version are created and inserted into the broadcast table. Next, a broadcast response is sent back to the server 104 (step 2583).

At step 2584, a close session request is sent to the server 104 and the communication is disconnected. The application download/update histories table (see Figure 11) and the application registration table (see Figure 7) are updated (step 2585). In an exemplary embodiment, a corresponding record in the application download/update histories table is updated as follows: the "appSize" field value is replaced with the application size of the updated application, the "nUpdate" field is incremented by 1, the "timestamp" field value is replaced by the current time, and the "updateRate" field value is recalculated. In an exemplary embodiment, new updateRate is equal to (the old updateRate x old nUpdate + diffSize x

100/appSize)/new nUpdate, where diffSize is the size difference between the new and old versions of the application. A corresponding record in the application registration table is updated as follows: the "appVer" filed is replaced with the version of the downloaded application. Next, the application cached in the gateway 108 is updated based on the at least one difference file (step 2586) and differences between the updated application and an earlier version of the application are generated. A new CBI is calculated for the updated application (step 2587). In an exemplary embodiment, the new CBI is equal to the old CBI plus the size difference between the versions at the gateway 108 and at the mobile device minus the size difference between the updated version and the original version at the gateway. In another exemplary embodiment, the "updateRate" field is also updated. In one embodiment, the new update rate is equal to ((old updateRate^{sg} x old nUpdate^{sg}) + diffSize x 100/ new appSize)/ new nUpdate^{sg}, where new appSize is the size of the updated application.

Next, the application storage table is updated (step 2588). In an exemplary embodiment, a record associated with the updated application in the application storage table is updated as follows: the "nFile," "fNames," "fVers," "nextRel," "lang," flagSet," "nUpdate," "updateRate," "CBI," and "updateItvl" fields are updated with new values. In an exemplary embodiment, a new updateItvl is equal to (1 - UPDATE_TM_WEIGHT) x old updateItvl + UPDATE_TM_WEIGHT x (timestamp$_{now}$ - lastUpdate), where UPDATE_TM_WEIGHT is as defined in Table 18, the lastUpdate is equal to the time of the last update, and the timestamp$_{now}$ is the current time. Next, the at least one difference file, which represents the difference between the updated version and the version cached at the mobile device 110, is loaded from the local file system 226 (step 2589), the associated meta information is loaded from the application storage table in the gateway database 218 (step 2590), and the process continues at step 2528 in Figure 25A.

Figures 26A-26C schematically illustrate three modes of exemplary application/data status check processes in accordance with an embodiment of the invention. In Figure 26A, the gateway 108 facilitates a direct application/data status check between a server 102-106 and the mobile device 110 without caching the application/data. In Figure 26B, a requested application/data is already cached at the gateway 108 and the application/data is up-to-date; the gateway 108 directly checks the status of the application/data at the mobile device 110 without contacting the

server 102-106. In Figure 26C, a requested application/data is already cached at the gateway 108 and the cached version is out-of-date; the gateway 108 performs a status check on the cached application/data with the server 102-106 then performs a status check on the application/data at the mobile device 110.

Figures 27A-D illustrate an exemplary application status check process in accordance with an embodiment of the invention. At step 2702, an application status check request is received. The application requested is authenticated (step 2704). In an exemplary embodiment, an application URL is identified from the application update request. The application identification table (see Figure 4) and the application registration table (see Figure 7) in the gateway database 218 are searched using the application URL. If a matching record exists and the associated application is not disabled (e.g., the flagSet field is in the application registration table "off"), then the application is authenticated. If the application is not authenticated, an error response is created (step 2708) and sent to the requesting mobile device 110. If the application is authenticated (step 2706), then whether the application is already cached is checked (step 2710). If the application is not cached (step 2712), whether a server, where the application can be downloaded, is a 3i server is determined (step 2714). If the server is a 3i server, the process continues in Figure 27B. Otherwise, the process continues in Figure 27C.

Referring back to step 2712, if the application is already cached, whether the application is up-to-date (or more current than the version cached at the mobile device 110) is determined (step 2716). If the application is not up-to-date (and/or is the same version as the version cached at the mobile device 110), the process continues in Figure 27D. If the application is up-to-date, the gateway database 218 is accessed to load broadcast information, if any (step 2718). A success response is generated (step 2720) and an application status check response is sent to the requesting mobile device 110 (step 2722). In an exemplary embodiment, the application status check response is generated by comparing the application version indicated in the status check request and the application version in the application registration table (see Figure 7) at the gateway database 218 or downloaded/generated from the server response.

Figure 27B illustrates an exemplary status check process when the server is a 3i server 104 in accordance with an embodiment of the invention. At step 2724, an open/reuse communication session request is sent to the 3i server 104. An open/reuse communication session response is received (step 2726). Next, an application status

check request is sent to the server 104 (step 2728) and an application status check response is received from the server 106 (step 2730). In an exemplary embodiment, the response includes the current version and status of the application. The response is parsed to determine whether a broadcast message is piggybacked (step 2732). If no broadcast message is piggybacked (step 2734), the process continues at step 2742. If a broadcast message is piggybacked (step 2734), the application storage table (see Figure 13) is accessed and updated (step 2736). In an exemplary embodiment, a broadcast message includes an application URL and an application version for each of one or more applications. The application storage table is searched for the *appVer* and *flagSet* fields of each record that is associated with an application URL component in the broadcast message. The *appVer* of a matching record and an application version component in the broadcast message are compared. If the versions are different, then set a corresponding *flagSet* field to indicate that the associated application is out-of-date. This process repeats for all applications in the broadcast message. Next, the mobile application cache table (see Figure 15) and the broadcast table (see Figure 17) are accessed (step 2738). In an exemplary embodiment, the mobile application cache table is searched for the "subId" fields of all records that are associated with the application URLs in the broadcast message. For every identified subId field, a record comprising the subId, the application URL, and the application version are created and inserted into the broadcast table. Next, a broadcast response is sent back to the 3i server 104 (step 2740). At step 2742, a close session request is sent to the server 104 and the communication is disconnected and the process continues at step 2718 in Figure 27A.

Figure 27C illustrates an exemplary status check process when the server is a non-3i server 102 or 106 in accordance with an embodiment of the invention. At step 2744, a connection with the server 102 is established. Next, an application download request is sent, a response is received, and an application is downloaded from the server 102 (step 2746). The communication is disconnected (step 2748). The application version is generated from the downloaded application by calling the version calculator 230 (see Figure 2) (step 2750) and the process continues at step 2718 in Figures 27A. In an exemplary embodiment, the version calculator 230 compares the downloaded application and the application already cached at the gateway 108 or at the mobile device 110.

Figure 27D illustrates another exemplary status check process when the server is a 3i server 104 in accordance with an embodiment of the invention. At step 2752, an open/reuse communication session request is sent to the 3i server 104. An open/reuse communication session response is received (step 2754). Next, an application status check request is sent to the server 104 (step 2756) and an application status check response is received from the server 104 (step 2758). In an exemplary embodiment, the response includes the current version and status of the application. The response is parsed to determine whether a broadcast message is piggybacked (step 2760). If no broadcast message is piggybacked (step 2762), the process continues at step 2770. If a broadcast message is piggybacked (step 2762), the application storage table (see Figure 13) is accessed and updated (step 2764). In an exemplary embodiment, a broadcast message includes an application URL and an application version for each of one or more applications. The application storage table is searched for the *appVer* and *flagSet* fields of each record that is associated with an application URL component in the broadcast message. The *appVer* of a matching record and an application version component in the broadcast message are compared. If the versions are different, then set a corresponding *flagSet* field to indicate that the associated application is out-of-date. This process repeats for all applications in the broadcast message. Next, the mobile application cache table (see Figure 15) and the broadcast table (see Figure 17) are accessed (step 2766). In an exemplary embodiment, the mobile application cache table is searched for the "subId" fields of all records that are associated with the application URLs in the broadcast message. For every identified subId field, a record comprising the subId, the application URL, and the application version are created and inserted into the broadcast table. Next, a broadcast response is sent back to the 3i server 104 (step 2768). At step 2770, a close session request is sent to the server 104 and the communication is disconnected and the process continues at step 2718 in Figure 27A.

In an exemplary embodiment, application and file versions are calculated by applying a one-way hashing function (e.g., MD4). To calculate an application version, all files belonging to an application are organized in a defined order then a one-way hashing function is applied to the files. This method assumes that servers generally download contents of an application in a consistent order. To calculate a file version, contents of a file is organized in a byte stream then a one-way hashing function is

applied to the byte stream. Generally, the same one-way hashing function is used for calculating both application and file versions.

Although Figures 21-27 illustrate exemplary processes to process applications, a person skilled in the art would recognize that these processes similarly apply to process data.

An application or data typically comprises a set of files. When an application or data is updated, one or more of a corresponding set of files is updated (i.e., added, modified, or removed). In an exemplary embodiment, one or more difference files are created by the gateway 108 or the 3i server 104 that represents the difference between the old version and the new version of the application to be updated. A difference file provides information regarding a file to be added to an original set of files, a file in an original set of files that should be modified, or a file in an original set of files that should be deleted. For example, for adding a file, a difference file includes the file's name, a 16-byte version information, contents of the new file, and the size of the new file in bytes. For deleting a file, a difference file includes the name of the file to be deleted. For modifying a file, a difference file includes a description of the difference between the modified file and the original file or the contents of the modified file, whichever is smaller.

Figure 28 illustrates an exemplary process to identify the difference between an old application and a new application in accordance with an embodiment of the invention. At step 2802, an old application and a new application are received. Each application comprises a set of files. Next, the files that were added into the new application are identified (step 2804). The files that were deleted from the new application are identified (step 2806). The files that were modified in the new application are identified (step 2808). The difference between each corresponding pair of changed files is calculated (step 2810). All calculated changes are bundled together (step 2812).

The smart connectivity protocol (SCP) is a protocol used for application/data management between the mobile device 110 and the gateway 108 or between the mobile device 110 and a remote server 102. Figure 29 illustrates exemplary state machines of the SCP in accordance with an embodiment of the invention. Generally, when the SCP is in an Idle state, no communication session is created and, thus, no communication activity is taking place. When the SCP is in an Open state, a communication session is created; the system may be for communication requests

from a client. When the SCP is in a Download state, a download request is sent or a download response is prepared. When the SCP is in an Update state, an update request is sent or an update response is prepared. When the SCP is in an Initialize state, an initialization request is sent or an initialization is prepared. When the SCP is in a Register state, cache changes are piggybacked or an acknowledgment is prepared. When the SCP is in a Broadcast state, broadcasts are piggybacked or an acknowledgment is prepared.

The foregoing examples illustrate certain exemplary embodiments of the invention from which other embodiments, variations, and modifications will be apparent to those skilled in the art. The invention should therefore not be limited to the particular embodiments discussed above, but rather is defined by the claims.